UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

19|20

Trabajo fin de grado

Herramienta interactiva para el estudio y visualización de paseos aleatorios

Jesús Miguel Álvarez Domínguez

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Herramienta interactiva para el estudio y visualización de paseos aleatorios

Autor: Jesús Miguel Álvarez Domínguez
Tutor: Simone Santini

julio 2020

Jesús Miguel Álvarez Domínguez
Herramienta interactiva para el estudio y visualización de paseos aleatorios

Jesús Miguel Álvarez Domínguez
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Hacemos sñáquete y ya estaría.

Matteo Bonforte

Agradecimientos

A mis padres por haberme apoyado durante todos estos años y haberme convertido en la persona que soy hoy día.

Al siebenundzwanzigs, habéis hecho de mis años en la universidad una de las épocas más bonitas de mi vida, en especial a mi rubio favorito y eterno compañero de prácticas. Os llevaré siempre conmigo.

A mi pareja, por aguantarme todos estos años y haber sido uno de mis grandes apoyos.

Y a mi tutor, por soportar todos mis mails y ayudarme con la elaboración de este proyecto. Y en general a todas las personas que han estado a mi lado durante este proceso.

Resumen

Los paseos aleatorios y la difusión de información son importantes herramientas matemáticas que sirven para entender como la información se propaga y cómo la estructura de los grafos a los que se aplican modifican el proceso de difusión. A pesar de haber sido ampliamente estudiados y usados por muchas ramas de conocimiento, es complicado entender de forma simple cómo funcionan realmente al aplicarlos sobre casos reales.

Este trabajo de fin de grado consiste en el desarrollo de una aplicación web intuitiva que permita crear grafos manual o aleatoriamente en los que poder seguir el funcionamiento de distintos algoritmos de difusión de información en tiempo real. Además, la estructura general del grafo puede modificarse durante el proceso de difusión por lo que será sencillo apreciar los cambios que ello conlleva en la distribución de información sobre los distintos nodos.

La aplicación tiene un enfoque principalmente didáctico ya que ha sido concebida como herramienta de estudio o soporte para la enseñanza, pero puede usarse también como herramienta básica para la investigación de la teoría matemática de grafos.

Palabras clave

Grafos, Javascript, difusión de información, visualización, paseos aleatorios, p5.js

Abstract

Random walks and diffusion of information are main mathematical tools that allow us to understand how information is spread throughout graphs and how their structure modifies the diffusion process. Even though they have been widely studied and used in many fields of knowledge, it is still complicated to understand intuitively how they really work when applied to some real cases.

This bachelor thesis consists in the development of a web application that allows to create graphs manually or randomly in which we can follow the behaviour of different diffusion algorithms in real time. Furthermore, the main structure of the graph can be modified during the process of diffusion so it can be easier to appreciate the changes those modifications entails in the distribution of information on the different nodes.

The application is mainly meant as a didactic tool as it has been developed as a study tool or as a teaching support, but it can also be used as a basic research tool in graph theory.

Keywords

Graphs, Javascript, information diffusion, visualization, random walks, p5.js

Índice

Listas

Lista de ecuaciones

Lista de figuras

1

Introducción

En este capítulo de la memoria voy a exponer una introducción general al proyecto desarrollado explicando la motivación que se ha seguido para desarrollar la herramienta y los objetivos principales de la misma.

1.1. Motivación

En muchos grados universitarios relacionados con las ciencias de la computación o las matemáticas existen asignaturas en las que se imparte teoría de grafos. Toda esta teoría surge de la motivación de modelizar procesos naturales que ocurren en el día a día, ya sea las relaciones sociales entre personas o simplemente relaciones de causa-efecto entre distintos procesos.

No siempre se dispone de herramientas que apoyen la explicación de ciertos conceptos matemáticos con visualización en tiempo real de los cambios que estos generan sobre los grafos, por lo que en muchos casos dichas explicaciones quedan vacías de contenido visual.

Sin embargo, las herramientas que pueden utilizarse en la docencia son muy limitadas y en muchos casos no se dispone de medios digitales específicos que ilustren las expicaciones, por lo que los programas que se usen durante una clase deben ser rápidos, visuales y sobre todo simples. Por ello, uno de los recursos más utilizados es la consulta de páginas web que contengan la información necesaria.

Actualmente existen algunas herramientas online enfocadas a la teoría de grafos, como por ejemplo *Graph Online* [1] centrada en los algoritmos de búsqueda de camino más corto; *Rhumbl* [2] que es capaz de generar distintos tipos de gráficas a partir de datos en tablas, muy útil en el caso de existir relaciones unidireccionales entre los nodos (ver figura 1.1); o *Gephi* [3] una herramienta más completa que permite la visualización de grafos a gran escala (ver figura 1.2)

Sin embargo, muchas de estas páginas web necesitan de suscripciones de pago o de registro para poder ser usadas, por lo que muchas veces no están al alcance de la comunidad docente para poder ser usadas durante una clase o para estudiantes que quieran practicar con ellas.

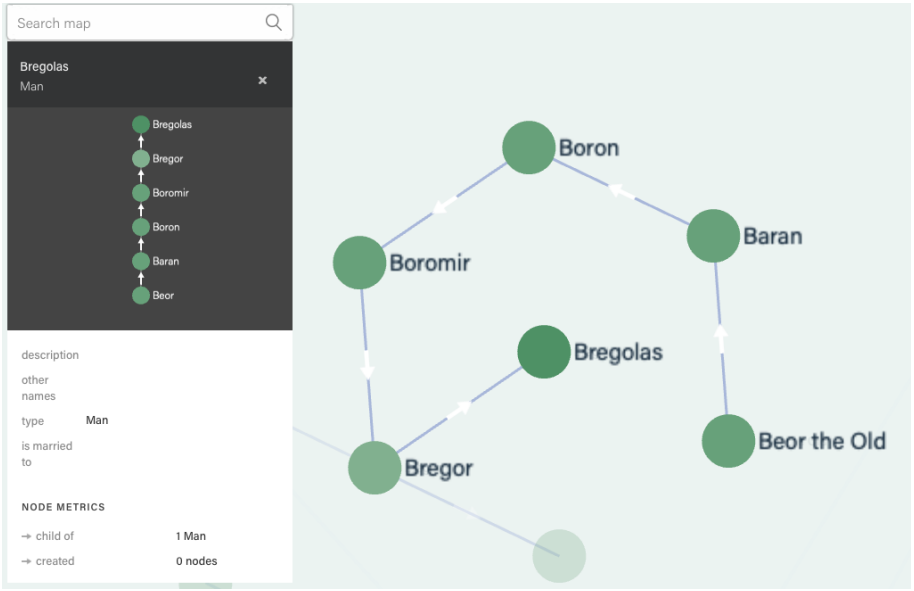


Figura 1.1: Ejemplo de uso de la herramienta Rhumbl para grafos con relaciones unidireccionales

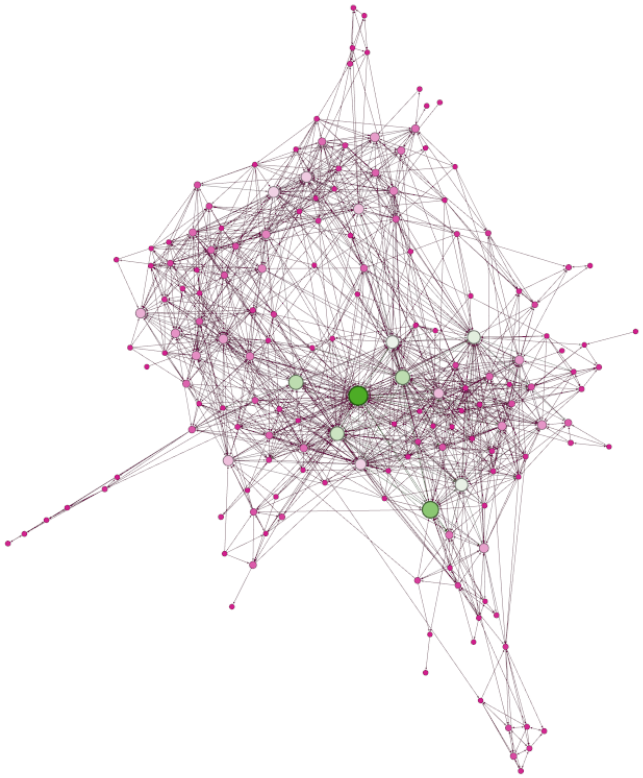


Figura 1.2: Ejemplo de uso de la herramienta Gephi para la visualización de un grafo a gran escala

Por otro lado, no se encuentran herramientas que se enfoquen en el problema de la difusión de información a través de un grafo o incluso del uso de algoritmos que recorran el grafo, como el algoritmo de paseos aleatorios, por lo que es complicado simular este tipo de procesos usando herramientas online.

Con todo ello, el objetivo principal del proyecto explicado a lo largo de esta memoria es la necesidad de una aplicación enfocada en la difusión de información en grafos no dirigidos con el propósito de poder reforzar los conocimientos de asignaturas como *Estructuras Discretas y Lógicas*, *Matemáticas Discretas* o cualquier otra cuyo temario incluya teoría de grafos. A pesar de estar enfocada al ambito docente, esta herramienta puede ser usada por todo usuario con intención de aprender sobre el funcionamiento de los algoritmos que se exponen en la memoria.

1.2. Objetivos

Objetivo 1 Crear una aplicación que sirva de apoyo a profesores y alumnos para estudiar y aprender sobre teoría de grafos y difusión de información.

Objetivo 2 Implementación de distintos algoritmos de difusión para poder ser aplicados sobre grafos no dirigidos.

Objetivo 3 Diseño de una interfaz gráfica visualmente atractiva y simple para permitir que los usuarios accedan a todas las opciones de manera rápida e intuitiva.

Objetivo 4 Implementación de un sistema que permita al usuario crear un grafo personalizado añadiendo y eliminando aristas o nodos.

2

Paseos Aleatorios y Difusión

Los algoritmos de difusión de información y paseos aleatorios han sido ampliamente estudiados en muchos campos tanto de la ciencia como de humanidades, desde la física (movimiento Browniano [4]) hasta la epidemiología [5] y la sociología [6]. Podemos dividir este estudio en dos grandes clases. Por un lado, existen los paseos aleatorios que se desarrollan en un continuo, que normalmente se asume ser R o R^2. En este caso los paseos se describen a nivel global con ecuaciones de tipo Fokker-Planck [7] que describe cómo evoluciona en el tiempo la función de densidad de la distribución probabilística de una partícula sometida a fuerzas aleatorias.

En otros casos, especialmente en el estudio de la búsqueda y difusión de información en redes sociales, se estudian paseos aleatorios en grafos considerando que el espacio del paseo es un conjunto finito o numerable de puntos o nodos. El paseo se lleva a cabo sólo a través de las aristas del grafo. En este caso, el estudio analítico es más complicado ya que si seguimos teniendo el concepto de variación temporal de la probabilidad, no podemos definir operadores differenciales espaciales, complicando así el estudio analítico de la difusión relacionada con estos paseos. En este trabajo nos interesamos en este último tipo de paseos en grafos.

Con todo ello, de ahora en adelante, consideraremos un grafo no dirigido G compuesto por el par $G = (V, E)$ donde V será un conjunto finito de vértices o nodos y $E \subseteq V \times V$ las diferentes aristas o arcos (en inglés *edges*) que representan una relación entre dichos nodos.

2.1. Paseos Aleatorios

Un paseo aleatorio en un grafo es un proceso estocástico que comienza en uno de los vértices $v \in V$. A cada unidad de tiempo t se escoge un vértice conectado a v a través de una arista y el algoritmo 'camina' al nuevo vértice para volver a aplicar el mismo criterio en $t + 1$. De esta forma, la posibilidad de moverse de un vértice a otro se escoge de forma uniformemente aleatoria entre los vecinos de v.

Normalmente en lugar de determinar en qué vértice se encuentra el algoritmo en una unidad de tiempo concreto de la simulación, se considera una distribución probabilística del paseo aleatorio, es

decir, la probabilidad de que el algoritmo se encuentre en un vértice determinado. Para ello, consideramos la función $p : V \to R$ donde $p(v)$ es la probabilidad de estar en el vértice v con la distribución p. Dicha función debe cumplir dos condiciones:

1.– $p(v) > 0 \; \forall v \in V$

2.– $\sum_{v \in V} p(v) = 1$

Consideramos también el vector $p_t \in R^n$ como la distribución de probabilidad en un tiempo t del algoritmo. Por tanto, si comenzamos el paseo aleatorio en el vértice v tendremos

$$p_0(w) = \begin{cases} 1 & \text{si } w = v \\ 0 & \text{si } w \neq v \end{cases} \tag{2.1}$$

Para saber cuál es la probabilidad en un tiempo $t+1$ del nodo u tendremos que usar $p_t(u)$, dado que dicha probabilidad vendrá dada por la suma de probabilidades de los vecinos de u multiplicado por la probabilidad de que el algoritmo se haya movido hacia u, por lo que tenemos

$$p_{t+1}(u) = \sum_{v:(v,u) \in E} p_t(v)/d(v) \tag{2.2}$$

donde $d(v)$ es el grado del nodo v, es decir, el número de nodos a los que está conectado. Podemos reunir todas estas probabilidades en un vector

$$P(t) = (p_t(0), \ldots, p_t(n))' \tag{2.3}$$

El estudio de la difusión de la probabilidad en el paseo se reduce por tanto al estudio de la evolución en el tiempo del vector P. Para este estudio, necesitaremos una caracterización del grafo en el que se lleva a cabo el camino.

2.2. Difusión de Información

En los algoritmos de difusión aplicados en grafos, debemos imaginar que cada uno de los nodos de G contiene una cantidad finita de información que en cada unidad de tiempo se va distribuyendo por el grafo. La distribución de la información usando paseos aleatorios seguirá la ecuación 2.2, lo que implica que la probabilidad de tener información en un nodo obedece la ecuación de difusión.

2.2.1. Matriz de Adyacencia

Para facilitar los cálculos necesarios para entender los paseos aleatorios y la difusión en grafos, usaremos álgebra lineal. Para ello, definimos una matriz cuadrada de dimensión $n \times n$ donde n es el

número de nodos en G. Las entradas de la matriz vendrán dadas por:

$$A(u,v) = \begin{cases} 1 & \text{si } (u,v) \in E \\ 0 & \text{si } (u,v) \notin E. \end{cases} \tag{2.4}$$

Dicha matriz la llamaremos *matriz de adyacencia*, y nos indica los nodos que están unidos por aristas en nuestro grafo G. Nótese que si el grafo es no dirigido, la matriz A será simétrica.

Definimos también la matriz diagonal D^{-1} definida por

$$D^{-1}(u,v) = \begin{cases} 1/d(u) & \text{si } u = v \\ 0 & \text{si } u \neq v. \end{cases} \tag{2.5}$$

Y finalmente, definimos la *matriz de paso* como $W = AD^{-1}$. El elemento W_{ij} representa la probabilidad de que el paseo se mueva del nodo i al nodo j en el siguiente paso. Para el algoritmo de paseos aleatorios vendrá dada por

$$W(u,v) = \begin{cases} 1/d(u) & \text{si } (u,v) \in E \\ 0 & \text{si } (u,v) \notin E. \end{cases} \tag{2.6}$$

Con esta matriz W, vemos que la ecuación 2.2 es equivalente a la multiplicación matricial

$$P(t+1) = W'P(t) \tag{2.7}$$

Por lo que para un tiempo t podemos calcular los valores del vector P usando los valores iniciales $P(0)$ de la siguiente forma:

$$P(t) = W'P(t-1) = (W')^2P(t-2) = \cdots = (W')^tP(0) \tag{2.8}$$

Ejemplo (Parte I): En el problema de los puentes de Königsberg, analizado por Leonhard Euler en 1741 [8], tenemos el grafo 2.1 cuyas matrices serán

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad D^{-1} = \begin{pmatrix} 1/3 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1/3 \end{pmatrix} \quad W = \begin{pmatrix} 0 & 1/3 & 1/3 & 1/3 \\ 1/2 & 0 & 0 & 1/2 \\ 1/2 & 0 & 0 & 1/2 \\ 1/3 & 1/3 & 1/3 & 0 \end{pmatrix}$$

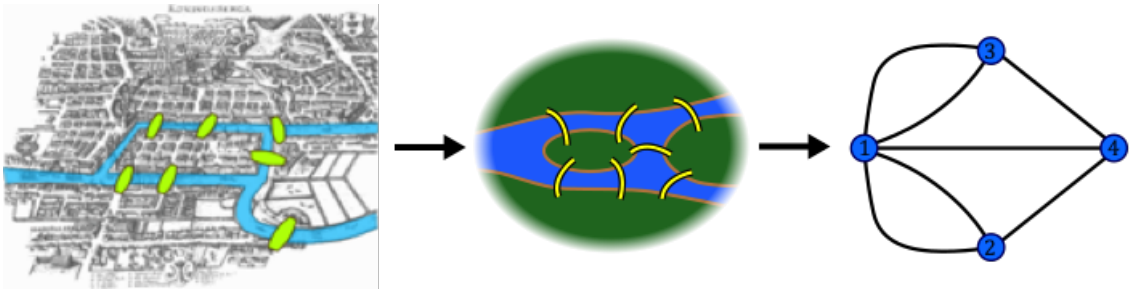


Figura 2.1: Creación de un grafo a partir del problema de los puentes de Königsberg

Estado Estable

Una de las cuestiones más importantes al valorar un algoritmo de difusión de información es la existencia o no de un estado estable, es decir, si el algoritmo llega a un punto de estabilidad para un período de tiempo largo. Considerando un vector de probabilidades π tal que

$$\pi_i = \frac{d(i)}{\sum_{j \neq i} d(j)}$$

es fácil ver que

$$W'\pi = \pi$$

donde π es el vector de probabilidad del estado estable en el grafo. Nótese que una vez que el grafo alcanza el estado π, éste permanecerá en dicho estado en los sucesivos períodos de tiempo.

Existencia de un Estado Estable

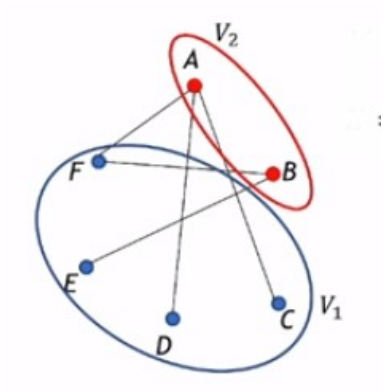


Figura 2.2: Ejemplo de grafo bipartito

El grafo contará con un estado estable siempre que no sea *bipartito* (ver figura 2.2), lo que implica que G puede ser dividido en dos subconjuntos V_1 y V_2 tal que $V_1 \cup V_2 = V$ y $V_1 \cap V_2 = \emptyset$ y tal que los

nodos de uno de los conjuntos solo tendrán uniones con los nodos de la otra parte.

En estos casos, el algoritmo llevará la información de uno de los subconjuntos al otro sucesivamente, por lo que nunca se llegará a un estado estable π.

Unicidad del Estado Estable

Lemma 2.2.1. *Sea $G = (V, E)$ un grafo conexo no dirigido y no bipartito y sea p un vector tal que $Wp = p$, entonces $p = \pi$*

Demostración. Tomamos $u \in V$ el vértice que maximiza

$$\frac{p(u)}{d(u)}$$

Según la hipótesis, tenemos que

$$\begin{aligned} p(u) &= \sum_{v:(u,v)\in E} \frac{p(v)}{d(v)} \\ &\leq \sum_{v:(u,v)\in E} \frac{p(u)}{d(u)} \\ &= p(u) \end{aligned}$$

Y la única forma de conseguir esa desigualdad es notando que de hecho la segunda linea es una igualdad, por lo que

$$\frac{p(v)}{d(v)} = \frac{p(u)}{d(u)} \tag{2.9}$$

para todos los vecinos v de u. Finalmente, dado que el grafo es conexo, tenemos que el ratio 2.9 se mantiene tanto para los vecinos directos de u como para los vecinos de segundo grado (vecinos de vecinos) y debido que existe un camino que une u con cualquier nodo del grafo, se cumple para todos los nodos del grafo. □

Obtención del Estado Estable

A pesar de que la matriz $W = AD^{-1}$ no es simétrica por lo general, podemos usar $D^{1/2}$ cuya n-diagonal está compuesta por $\sqrt{d(n)}$; y $D^{-1/2}$ con $1/\sqrt{d(n)}$ en la diagonal correspondiente. Con ello, construimos para el algoritmo de paseos aleatorios la matriz diagonal M que es *similar* a W:

$$M = D^{-1/2}WD^{1/2} = D^{-1/2}(AD^{-1})D^{1/2} = D^{-1/2}AD^{-1/2} \tag{2.10}$$

Lo primero que observamos es que M y W tienen los mismos autovalores por similaridad. Además para cada autovector v de M y su correspondiente autovalor λ, tenemos

$$\lambda v = Mv = D^{-1/2}WD^{1/2}$$
$$\lambda(D^{1/2}v) = W(D^{1/2}v)$$

por lo que $(D^{1/2}v)$ es un autovector de W. Si ahora elevamos la matriz M por t tenemos:

$$W^t x = (D^{1/2}MD^{-1/2})^t x = D^{1/2}M^t D^{-1/2}x = \sum_i \lambda_i^t D^{1/2} v_i (v_i^T D^{-1/2} x).$$

Por lo que lo único que cambia a medida que t crece es la potencia de los autovalores. Aquellos autovalores con valor absoluto menor que 1 tenderán a 0 y solo quedarán los autovalores 1 y -1.

Ejemplo (Parte II): Siguiendo el problema de los puentes de Königsberg, construimos la matriz M como

$$M = D^{-1/2}AD^{-1/2} =$$
$$= \begin{pmatrix} 1/\sqrt{3} & 0 & 0 & 0 \\ 0 & 1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 & 1/\sqrt{3} \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1/\sqrt{3} & 0 & 0 & 0 \\ 0 & 1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 & 1/\sqrt{3} \end{pmatrix} =$$
$$= \begin{pmatrix} 0 & 1/\sqrt{6} & 1/\sqrt{6} & 1/3 \\ 1/\sqrt{6} & 0 & 0 & 1/\sqrt{6} \\ 1/\sqrt{6} & 0 & 0 & 1/\sqrt{6} \\ 1/3 & 1/\sqrt{6} & 1/\sqrt{6} & 0 \end{pmatrix}$$

cuyos autovectores son

$$\lambda_1 = 1 \qquad \to v_1^T = (1, \sqrt{2/3}, \sqrt{2/3}, 1)$$
$$\lambda_2 = -2/3 \qquad \to v_2^T = (1, -\sqrt{3/2}, -\sqrt{3/2}, 1)$$
$$\lambda_3 = -1/3 \qquad \to v_3^T = (-1, 0, 0, 1)$$
$$\lambda_4 = 0 \qquad \to v_4^T = (0, -1, 1, 0)$$

Ahora que hemos identificado el autovector de M correspondiente al autovalor $\lambda_1 = 1$, aplicamos la transformación para conseguir el autovector de W:

$$v_1^W = D^{1/2}v_1 =$$
$$= D^{1/2}(1, \sqrt{2/3}, \sqrt{2/3}, 1)^T =$$
$$= \left(\sqrt{3}, \frac{2\sqrt{3}}{3}, \frac{2\sqrt{3}}{3}, \sqrt{3}\right)^T$$

Por lo que el estado estable del problema será el dado por v_1^W. Sin embargo, para entender mejor la proporción de información en cada nodo, se suele dividir este autovector por la suma de sus términos, de esta forma tendremos el porcentaje total de información en cada nodo:

$$\frac{v_1^W}{\text{sum}(v_1^W)} = \frac{\left(\sqrt{3}, \frac{2\sqrt{3}}{3}, \frac{2\sqrt{3}}{3}, \sqrt{3}\right)^T}{\frac{10\sqrt{3}}{3}} = \\ = \left(\frac{3}{10}, \frac{2}{10}, \frac{2}{10}, \frac{3}{10}\right)^T$$

Llegamos a la conclusión de que los nodos 1 y 4 contendrán un $30\,\%$ de la información total en el grafo si dejamos correr el algoritmo random walk durante un tiempo lo suficientemente largo; mientras que los nodos 2 y 3 contendrán un $20\,\%$ de dicha información.

2.3. Algoritmos de Difusión

Los algoritmos de difusión son una serie de procesos que modifican los valores de la matriz de paso. A partir de ahí, a cada unidad de tiempo se distribuirá la información que contiene el nodo de forma proporcional a cada entrada de su fila correspondiente. Existen diferentes criterios para esta matriz de paso, en esta sección exploraremos aquellos algortimos implementados en la herramienta.

Consideraré durante toda esta sección el grafo de ejemplo 2.3.

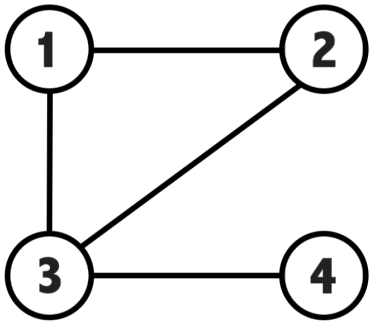


Figura 2.3: Grafo de Ejemplo

2.3.1. Random Walk

El primero de los algoritmos es una modificación de el algoritmo de paseos *Random Walk* o *Paseos Aleatorios*.

Si consideramos el caso discreto, a cada unidad de tiempo t una partícula que se encuentre en un nodo $v \in V$ elegirá aleatoriamente un nodo vecino de v y se moverá a dicho nodo. Sin embargo, en

los algoritmos de difusión se considera la información como algo parecido a un líquido o gas que se mueve por el grafo de manera continua. Por tanto, esta información se moverá de forma uniforme a través de los nodos vecinos del grafo. Con todo esto, la probabilidad de difusión en un nodo con grado $deg(v)$ será

$$\mathrm{P}(v \to u) = \frac{1}{deg(v)} \tag{2.11}$$

Ejemplo: Considerando el grafo 2.3 y aplicando el algoritmo Random Walk, tendríamos las probabilidades mostradas en la figura 2.4 y cuya matriz de paso será:

$$W = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 1/3 & 1/3 & 0 & 1/3 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

En la implementación de este método la diagonal de la matriz W contendrá el número -10 para identificar que la matriz se corresponde al algoritmo random walk.

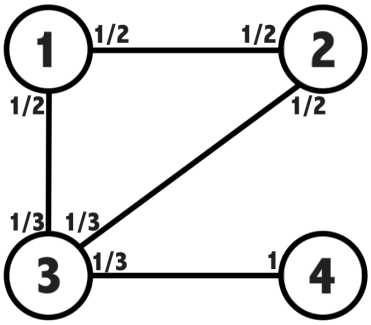


Figura 2.4: Aplicación algoritmo random walk

2.3.2. Lazy Random Walk

Otra versión del algoritmo random walk mencionada en [9] se denomina *Lazy Random Walk*. En este caso, solo se distribuirá la mitad de la información disponible en los nodos, por lo que la otra mitad quedará anclada en el nodo v como podemos ver en 2.5. La probabilidad vendrá dada por

$$\mathrm{P}(v \to u) = \frac{0{,}5}{deg(v)} \quad \text{para } u \neq v$$

Ejemplo: La matriz de paso del algoritmo lazy random walk quedará como

$$W = \begin{pmatrix} 1/2 & 1/4 & 1/4 & 0 \\ 1/4 & 1/2 & 1/4 & 0 \\ 1/6 & 1/6 & 1/2 & 1/6 \\ 0 & 0 & 1/2 & 1/2 \end{pmatrix}.$$

De hecho, notar que la diagonal de la matriz de paso W no sería necesaria en términos de computación, ya que de suprimirse, solamente la mitad de la información se transmitiría a los vecinos sin importar que se especifique explícitamente que la mitad de dicha información se mantiene en el nodo. De hecho, en la implementación de este método la diagonal contiene el número -20 para informar que se corresponde a la matriz de paso del algoritmo lazy random walk.

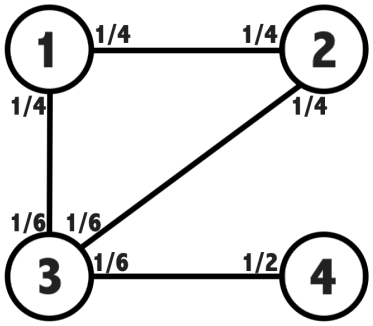


Figura 2.5: Aplicación algoritmo lazy random walk

2.3.3. Preferencial

Para simular el comportamiento que puede tener un conjunto de personas con relaciones bidireccionales se suele usar el algoritmo de difusión *Preferencial*. En este algoritmo los nodos con mayor número de vecinos o más "populares" serán los sumideros de información.

Para conseguir esto, cada nodo debe mirar al conjunto de sus vecinos y cacular la suma de sus grados, para después asignar las probabilidades proporcionales a cada vecino, es decir,

$$\mathrm{P}(v \to u) = \frac{deg(u)}{\sum_{w:(v,w)\in E} deg(w)}$$

Ejemplo: En el caso del algoritmo preferencial tendríamos la matriz correspondiente al grafo:

$$W = \begin{pmatrix} 0 & 2/5 & 3/5 & 0 \\ 2/5 & 0 & 3/5 & 0 \\ 2/5 & 2/5 & 0 & 1/5 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

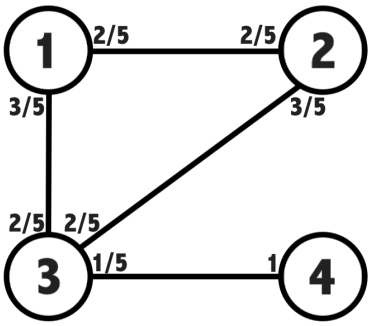


Figura 2.6: Aplicación algoritmo preferencial

En el caso de aplicación de este algoritmo, se usará un -30 en la diagonal para indicar que la matriz de paso W corresponde al algoritmo preferencial.

2.3.4. Page Rank

El último de los algoritmos usados en el proyecto se denomina *Page Rank* que fue desarrollado por Google en 1999 para asignar la relevancia de documentos o páginas web en su motor de búsqueda [10]. El algoritmo tiene un componente de aletoriedad, ya que distribuye la mayor parte de la información contenida en un nodo de forma uniforme entre sus vecinos y una pequeña parte de esta información la distribuye aleatoriamente entre los nodos del grafo con los que no comparte ninguna arista.

En la implementación de este algoritmo he decidido distribuir el $90\,\%$ de la información entre los vecinos y el $10\,\%$ enviarla a un solo nodo aleatorio no conectado, por lo que la fórmula quedaría como:

$$\mathrm{P}(v \to u) = \begin{cases} \frac{0{,}9}{deg(u)} & \text{si } (u,v) \in E \\ 0{,}1 & \text{si } (u,v) \notin E \quad \text{y} \quad u = \mathrm{Rand}(\{w \in V : (v,w) \notin E\}) \end{cases}$$

Ejemplo: Dada la aletoriedad del algoritmo, este es un ejemplo de la aplicación del page rank.

$$W = \begin{pmatrix} 0 & 0{,}45 & 0{,}45 & 0{,}1 \\ 0{,}45 & 0 & 0{,}45 & 0{,}1 \\ 1/3 & 1/3 & 0 & 1/3 \\ 0{,}1 & 0 & 0{,}9 & 0 \end{pmatrix}.$$

En el caso de tener un nodo “totalmente conectado”, ie. conectado con todos los demás nodos del grafo, no se reparte información de forma aleatoria, sino que se reparte el $100\,\%$ de forma uniforme (ver el nodo 3 de la figura 2.7). En la implementación del page rank, se completará la diagonal de la matriz del número -40 para indicar que se ha aplicado el algoritmo.

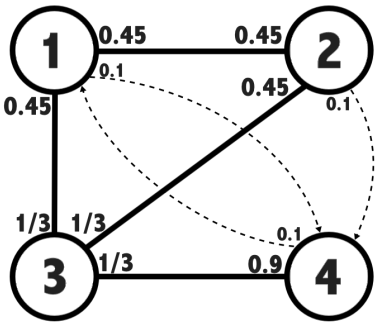


Figura 2.7: Aplicación algoritmo page rank

2.4. Generación de Grafos

Obviamente uno de los objetivos principales del proyecto es la visualización de los algoritmos actuando sobre grafos de distinta índole: con mayor o menor número de nodos, más conexiones, estructuras particulares, etc... Por ello, la creación de los grafos es una pieza clave.

2.4.1. Creación manual

Existe la posibilidad de que el usuario sea el que configure la forma del grafo añadiendo nodos y creando aristas entre ellos de forma manual. También existe la opción de limpiar el grafo en cualquier momento y empezar de nuevo. En la primera ejecución del programa, se creará un grafo de 5 nodos conectados entre sí de forma aleatoria, cada uno de ellos con una unidad de información.

2.4.2. Modelo Barabási–Albert

Una de las opciones disponible para el usuario es añadir un número de nodos personalizado siguiendo el modelo *Barabási-Albert* [11] que usa el mecanismo de *conexión preferencial*.

En este proceso, se irán añadiendo los nodos de uno en uno, cada uno de ellos sin ningún tipo de información. Sin embargo, las conexiones mediante aristas que se le otorgan a estos nuevos nodos vendrán dadas por una distribución de probabilidad proporcional al grado de los nodos a los que se conectan ya existentes en el grafo, es decir, que cuanto mayor sea el grado de un nodo mayor será la probabilidad de que se genere una arista desde el nodo que estamos creando mediante el modelo Barabási-Albert. Esta probabilidad se calcula mediante la ecuación:

$$\mathrm{P}((u,v) \in E) = \frac{\deg(v)}{\sum_{w \in V} \deg(w)} \tag{2.12}$$

siendo $P((u, v) \in E)$ la probabilidad de que un nuevo nodo u se conecte a un nodo v ya existente en el grafo. Por tanto, los nodos con mayor número de conexiones mediante aristas ("hubs") tienden a acumular un mayor número de conexiones nuevas, mientras que aquellos con menor número de conexiones tendrán menor probabilidad de crear nuevas aristas.

Se dice que estas redes tienen una distribución de grado que sigue una ley exponencial, es decir, que la fracción de nodos con un cierto grado k aumenta de forma exponencial a medida que vamos añadiendo nodos que sigan este modelo de conectividad preferencial.

3

Desarrollo de Algoritmos

Para enteder mejor cómo funcionan ciertos algoritmos es muy útil disponer de herramientas gráficas que nos faciliten su comprensión y estudio. Con todo ello, ser capaz de ver cómo funcionan los algoritmos de difusión con ejemplos prácticos y personalizados por el usuario es la mejor forma de experimentar y de entenderlos.

3.1. Visualización

Para mayor alcance de la herramienta, ésta ha sido implementada para ser ejecutada en un entorno web. Con ese motivo, he usado entornos gráficos que permiten desarrollar y dibujar todas las partes necesarias.

3.1.1. Visualización de grafos basada en vínculos elásticos

Los algoritmos de fuerza para grafos son una manera de hacer que los nodos y vértices se dibujen de una manera más estética y ayudan a la mejor comprensión de la estructura del grafo.

Su principal objetivo es dotar a los nodos de una fuerza de repulsión que haga que aquellos que se encuentran próximos se repelan mutuamente. Para simular dicha fuerza suele utilizarse fórmulas basadas en la Ley de Coulomb, que define la fuerza de repulsión entre partículas eléctricas con cargas del mismo signo.

$$F_a = \kappa \frac{q_1 q_2}{r^2} \tag{3.1}$$

donde κ es la constante de Coulomb, q_i la carga de las partículas y r^2 la distancia al cuadrado entre dichas partículas. En la implementación del algoritmo de fuerza, se han suprimido los valores de las cargas electrostáticas dado que se considera a todos los nodos con la misma fuerza de repulsión. Por lo que la fuerza de repulsión será directamente proporcional a una constante `C_0` e inversamente proporcional a la distancia al cuadrado entre los dos nodos.

Además de esta fuerza de repulsión, se les aplica a las aristas del grafo una fuerza de atracción elástica simulando el comportamiento de muelles. Para ello, se usan fórmulas basadas en la Ley de elasticidad de Hooke, que nos permiten calcular con que fuerza interactuan dos partículas unidas mediante una arista. Al ser un muelle existe una longitud que se considera estable, a mayor o menor longitud de la arista, ésta actuará atrayendo o repeliendo los nodos respectivamente hasta volver a su longitud estable.

$$F_r = -k\delta \tag{3.2}$$

donde k es la constante de elastica del muelle y δ la variación de longitud con respecto a su estado estable. En la implementación, k viene dada por la constante `C1` y la elongación estable de las aristas viene dada por la constante `SPRING.LENGTH`

Código 3.1: Esta función se encarga de aplicar las físicas necesarias para que el grafo se disponga en la pantalla de forma estética

```
1   for (let node of this.nodeList) {
2     let force = createVector(0,0);
3
4     for (let otherNode of this.nodeList) {
5       if (node != otherNode) {
6         // Vector between the 2 nodes
7         let v = createVector(node.pos.x -otherNode.pos.x, node.pos.y -otherNode.pos.y);
8
9         // Repulsive force for nodes in the graph
10        var st = C0 / v.magSq();
11        let repulsive_force = v.normalize().mult(st);
12        force.add(repulsive_force);
13
14        // Attractive force for connected nodes
15        if (node.checkAdjacentNode(otherNode.id)) {
16          st = (-1) *C1 *(v.mag() -SPRING_LENGHT);
17          let attractive_force = v.normalize().mult(st);
18          force.add(attractive_force);
19        }
20      }
21    }
22    node.applyForce(force);
23    node.updatePos();
24  }
```

Normalmente en este tipo de algoritmos suele añadirse también una fuerza atractiva hacia el centro del espacio donde se dibuja el grafo. En este caso no ha sido necesario, ya que la herramienta dispone de un espacio limitado en el cual los nodos están obligados a permanecer en su interior, por lo que no es necesario atraerlos hacia el centro.

3.1.2. Color de los nodos

Los nodos son coloreados de un rojo más o menos intenso cuanta mayor o menor sea la cantidad de información que almacenan en su interior. Con ello se consigue que sea muy fácil identificar los nodos que según el algoritmo que corresponda tienden a ser sumideros de información. La función que se encarga de calcular la intensidad del color se consigue con una modificación de la función $\mathrm{erf}(x)$ y viene dada por la siguiente fórmula:

$$\mathrm{intensity} = 0{,}5 * (\mathrm{erf}(\mathrm{curvature} * (\mathrm{x} - \mathrm{mean})) + 1) \tag{3.3}$$

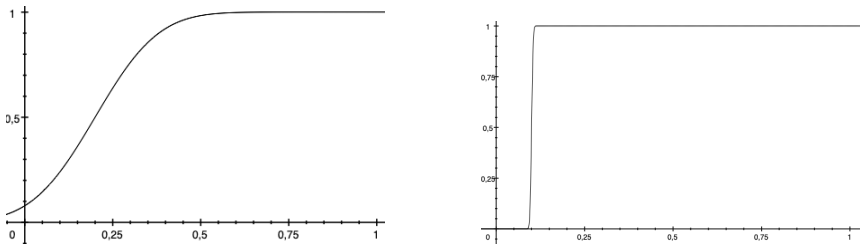
donde x es la proporción de información que tiene el nodo con respecto a la información total existente en el grafo, es decir

$$\mathrm{x} = \frac{\texttt{node.info}}{\texttt{graph.info}}$$

Los términos $curvature$ y $mean$ son los que configuran la forma específica de la curva. El problema de esta fórmula es que si queremos que se adapte al número de nodos o a la cantidad de información que discurre en el grafo tenemos que realizar cambios en la función $intensity$ en tiempo de ejecución. Para ello, tenemos

$$\mathrm{curvature} = \texttt{graph.n\_nodes}$$

esto es, la curvatura vendrá dada por el número total de nodos que haya en el grafo. A mayor número de nodos la pendiente de la curva será más cerrada y cuantos menos nodos más suave.



(a) Función intensidad con 5 nodos (b) Función intensidad con 200 nodos

Figura 3.1: A mayor número de nodos el cambio entre intensidad será más pronunciado, dado que así es más rápido localizar aquellos nodos con mayor información.

Esto permitirá que en un grafo con pocos nodos (menos de 15 aproximadamente) la diferencia entre intensidades permita al usuario percibir pequeñas variaciones en cuanto a la cantidad de información. Por otro lado, en grafos con un gran número de nodos es más útil marcar de forma muy intensa aquellos nodos con más información sin capturar pequeñas variaciones. Con ello, el usuario percibe más rápidamente los sumideros de información.

Por otro lado, tenemos la variable $\mathrm{mean} \in (0, 1/10]$, que moverá toda la gráfica hacia derecha o izquierda dependiendo de si su valor es más grande o más pequeño, respectivamente. Es una variable que ayuda a seleccionar mejor la media de información en el grafo y pintar los nodos que superen dicho valor. Se calcula como

$$\mathrm{mean} = \min\left(\frac{\texttt{node.info}}{\texttt{graph.info}}, 1/10\right)$$

esto es la información media que le correspondería a cada nodo, es decir, la división entre la información total y el número de nodos.

Tras calcular la intensidad correspondiente al nodo, necesitamos hacer la transformación

$$\mathrm{color} = 255 * (1 - \mathrm{intensity})$$

que cambia la función creciente por una decreciente, y la acota entre 0 y 255. Con ello, considerando el vector dado por $\mathrm{rgb} = (255, \mathrm{color}, \mathrm{color})$ tendríamos el color en formato RGB que pintaría de un rojo más intenso cuanta mayor cantidad de información contenga un nodo.

Nodos Barabási-Albert

Los nodos añadidos siguiendo el modelo Barabási-Albert de conexión preferencial [11] seguirán las mismas reglas descritas en esta sección para calcular la intensidad de color. Sin embargo, para hacer una distinción, serán dibujados de un verde más o menos intenso, siguiendo el vector dado por

$$\mathrm{rgb} = (\mathrm{color}, 255, \mathrm{color})$$

4

Diseño

A lo largo de este capítulo se expondrán los requisitos que el sistema cumple como parte de las decisiones de diseño que he tomado en la creación del proyecto. También se explicarán las implementaciones que he llevado a cabo para asegurar que se cumplen los requisitos. Por último, hablaré del diseño visual, funcionalidades y estructura general del proyecto.

4.1. Requisitos

Los requisitos del sistema se pueden dividir en: *requisitos funcionales*, que son aquellos que describen los servicios y funcionalidades que aporta la herramienta; y *requisitos no funcionales*, que nos aportan características de rendimiento y accesibilidad.

4.1.1. Requisitos Funcionales

RF-1: La herramienta debe permitir visualizar un grafo ya sea construido por el usuario o generado aleatoriamente.

RF-2: La herramienta debe mostrar la información de cada nodo, incluyendo la información contenida y su grado.

RF-3: La herramienta debe mostrar las aristas que unen los nodos en forma de línea.

RF-4: La herramienta debe permitir cambiar la información de cada nodo según la elección del usuario.

RF-5: La herramienta debe asegurar que el usuario no añade información negativa en un nodo.

RF-6: La herramienta debe mostrar la información global del grafo: grado medio, cantidad de información y el número de nodos y aristas

RF-7: La herramienta debe permitir modificar la estructura del grafo, bien añadiendo nodos o aristas, bien eliminándolos.

RF-8: La herramienta debe permitir al usuario borrar por completo el grafo actual y limpiar

toda la información del mismo.

RF-9: La herramienta debe ser capaz de activar o desactivar los algoritmos que permitan una distribución de nodos de forma estética para mejor visualización del grafo.

RF-10: La herramienta debe permitir al usuario elegir un algoritmo de difusión de información.

RF-11: La herramienta debe permitir modificar el ritmo del algoritmo elegido pausándolo, reactivándolo o acelerándolo.

RF-12: La herramienta debe ser capaz de restaurar el grafo previo a la ejecución de un algoritmo.

RF-13: La herramienta debe permitir que el usuario añada un número de nodos que seguirán un modelo de conexión preferencial.

RF-14: La herramienta debe asegurarse de que el usuario no añada más de 100 nodos con el modelo de conexión preferencial.

RF-15: La herramienta debe ejecutarse en un navegador web.

4.1.2. Requisitos No Funcionales

RNF-1: La herramienta debe ser intuitiva para que cualquier usuario pueda utilizarla de forma cómoda y segura.

RNF-2: La herramienta debe tener una interfaz sencilla para poder ser usada por usuarios con distinto nivel de conocimiento sobre teoría de grafos.

RNF-3: La herramienta debe permitir tener grafos con un gran número de nodos sin que ello suponga una ralentización del sistema.

RNF-4: La herramienta debe ejecutar los algoritmos de difusión de forma rápida con un tiempo de respuesta no superior a un segundo por iteración.

4.2. Tecnologías

En esta sección explico las distintas tecnologías usadas para la creación del sistema.

4.2.1. Javascript

El lenguaje de programación usado para el desarrollo será *Javascript* [12] ya que éste permite el desarrollo web. Una de las grandes ventajas es que es un lenguaje orientado a objetos por lo que podré almacenar todos los datos y variables de forma organizada mediante clases y relaciones entre clases. Además, dado que es un lenguaje ampliamente usado, dispondré de mucha información gracias a la

amplia comunidad de usuarios y gracias a la gran cantidad de librerias externas disponibles para el mismo.

4.2.2. p5.js

p5.js es una librería gratuita y open-source para Javascript [13] que contiene muchas herramientas para el dibujo en navegadores html. Con esta libreria he sido capaz de crear toda la componente gráfica de la herramienta. Además la comunidad de usuarios y colaboradores ha creado ejemplos de uso de todas las funciones, por lo que es sencillo aprender a usarla.

4.3. Estructura

En esta sección describo la estructura interna del programa, es decir, cómo ha sido pensado y construido. He seguido el patrón de diseño *MVC* o *Modelo-Vista-Controlador* [14] que se basa principalmente en separar las estructuras de datos de un programa, su representación visual y el conjunto de procesos que reaccionan a eventos. Dichas separaciones reciben el nombre de modelo, vista y controlador, respectivamente. Por ello, separaré la explicación en dichos módulos.

4.3.1. Modelo

Al usar un lenguaje orientado a objetos como *javascript* para la implementación, he creado las distintas clases que almacenan los datos internos del proyecto.

node.js

En esta clase he descrito la estructura interna y los métodos necesarios para la implementación de los nodos del grafo. Por una parte las propiedades de estos nodos abarcan desde la información que contiene, un identificador único o el conjunto de aristas que lo unen a otros nodos, así como las propiedades necesarias para que los algoritmos de visualización puedan dibujar el nodo correctamente: fuerza de desplazamiento, color del mismo, estado actual... etc. Además de los métodos de dibujado específicos del nodo y aquellos que modifican sus datos.

edge.js

En esta clase se encuentra la información de las aristas, que solamente contienen una referencia a los dos nodos que unen y un listado de flags que ayudan en el dibujado de las lineas. En cuanto a métodos, solo encontramos el método de dibujado específico para dibujar la línea entre los dos nodos correctamente.

graph.js

Esta es la clase principal del modelo de la herramienta, en ella se almacena toda la información correspondiente al grafo: nodos, aristas, matriz de paso y algoritmos de visualización. Además de la información global del grafo y también listas ordenadas de los nodos y aristas que lo forman.

En cuanto a métodos, encontramos primeramente aquellos que modifican la estructura del grafo ya sea añadiendo o eliminando nodos o uniéndolos mediante aristas. Por otra parte, la principal función de dibujado del grafo, que llama a las funciones específicas de nodos y aristas de forma que quede perfectamente dibujado. Y por último el algoritmo de fuerza explicado en la sección 3.1.1 que permite recolocar y mover los nodos para ayudar en la visualización de los mismos.

4.3.2. Vista

La vista del proyecto se compone de todos los métodos necesarios para generar una interfaz gráfica atractiva para el usuario y de ofrecerle todos los botones y elementos interactivos para manejar el programa. Todo ello se encuentra en el archivo *sketch.js*. Para realizar esta interfaz de usuario se ha usado la herramienta descrita en 4.2.2.

En la función *setup()* del archivo, se inicializan todos los paneles, botones y distintas vistas que tiene la aplicación en sí. A partir de ahí, se llama a la función *draw()*, que se ejecuta en cada frame y que dibujará tanto a pantalla principal donde se encuentra el grafo, como el panel que corresponda en ese momento.

4.3.3. Controlador

Por último, tenemos los métodos y funciones que se encargan de responder a la interacción del usuario con la interfaz gráfica, en otras palabras, aquellas funciones que usan la interfaz de usuario dada por la vista para que éste interactúe con los distintos elementos y modifican el modelo acorde a dichas decisiones. Para ello, encontramos tres tipos de métodos:

- Primeramente, cada uno de los botones y selectores que existen en el programa, han sido inicializados con funciones anónimas que ejecutan ciertos procesos al ser pulsados. Por ejemplo, el botón de *"Bloquear Grafo"* recorre todos los nodos de la estructura del grafo y cambia un flag específico para que en el siguiente frame en el que se ejecute el algoritmo de fuerza, éste deje de ejecutarse y no modifique la posición de los nodos.
- También existen funciones que reaccionan a los toques del ratón, tanto cuando se pulse, cuando se arrastre o cuando se libere.
- Por último, tenemos un conjunto de métodos que esconden o muestran uno de los tres paneles disponibles: Información, Control y Difusión dependiendo de la elección del usuario.

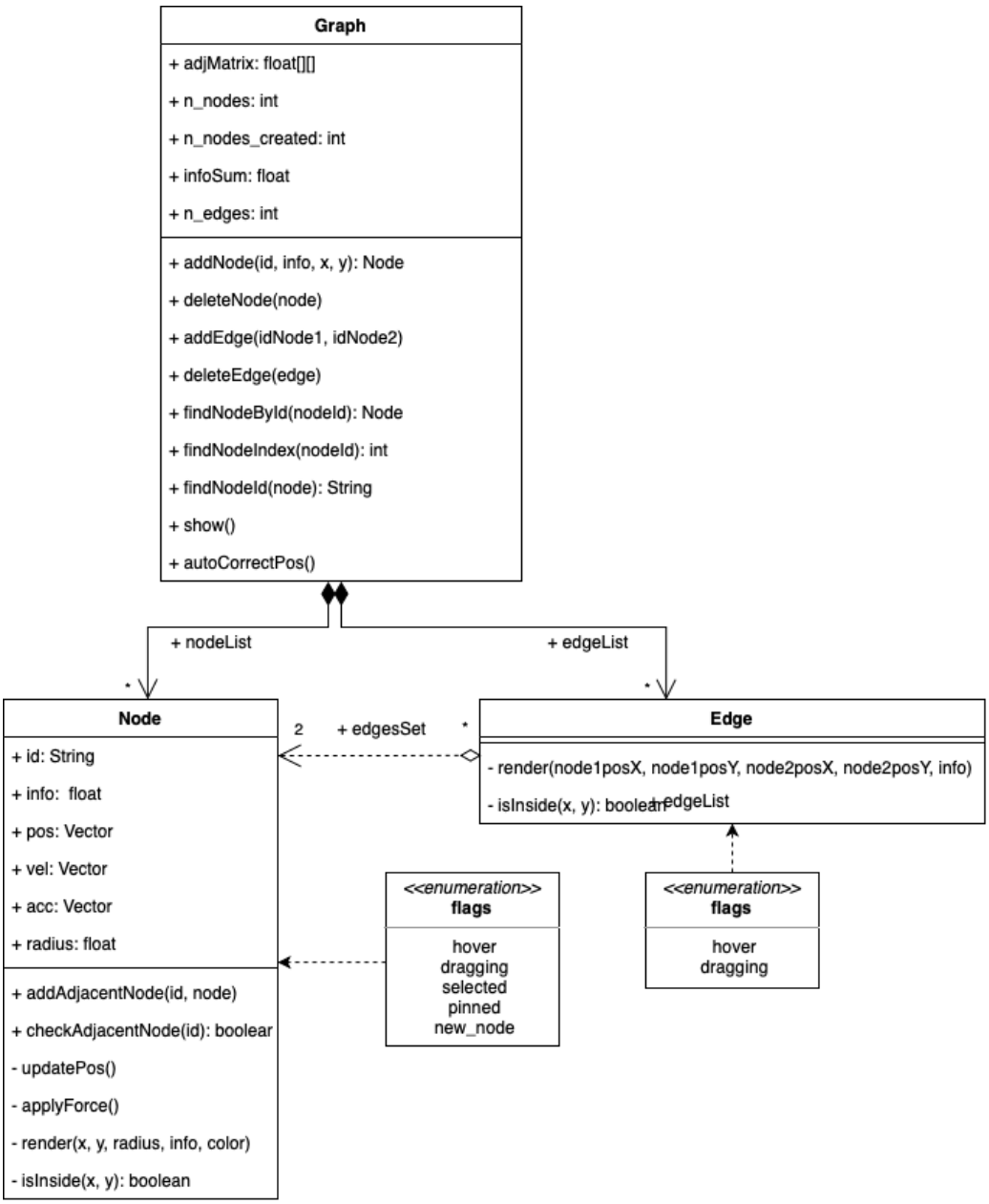


Figura 4.1: Diagrama de Clases del proyecto

4.3.4. Diagrama de clases

Como se puede apreciar en el diagrama 4.1, existen 3 clases en el proyecto. Su estructura interna ya ha sido explicada en la sección 4.3.1.

En cuanto a las relaciones entre ellas, como ya se introdujo anteriormente, la clase *Grafo* es la principal estructura del programa. En ella, observamos que hay tanto una lista de nodos (*nodeList*) como de aristas (*edgeList*) por lo que en ella encontramos toda la información útil de la herramienta, además de los métodos que modifican ambas estructuras.

Por otro lado, cada Nodo contiene una lista sin repetición de las aristas que le unen al resto de nodos del grafo, llamada *edgesSet*. Y cada *Arista* tiene en su estructura la referencia a los dos nodos que une.

Por último, las dos enumeraciones tanto de *Nodo* como de Arista sirven como flags que dan información sobre el estado de los elementos a los que están asociadas. Por ejemplo, si estamos moviendo con el ratón un nodo `node`, el flag `n.flags.dragging` será igual a `true`.

4.4. Interfaz de la aplicación

A continuación voy a mostrar las distintas vistas y paneles de la aplicación, explicando los distintos elementos y opciones disponibles para el usuario.

Para realizar el cambio entre los distintos paneles, se dispone en la esquina superior derecha de tres botones a modo de selección.

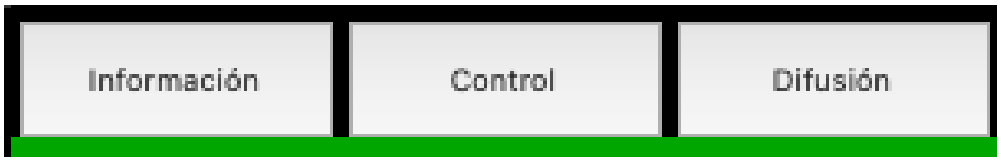


Figura 4.2: Botones de Selección de Panel

4.4.1. Panel del grafo

El único elemento estático de toda la herramienta y que por tanto será siempre visible será la vista del grafo en la que se muestra el grafo en todo momento.

Este grafo es en cierta forma interactivo para el usuario, ya que éste podrá arrastrar y mover nodos usando el ratón. Además, tal y como se menciona en la sección 2.4, se podrá crear un grafo personalizado usando todas las herramientas disponibles.

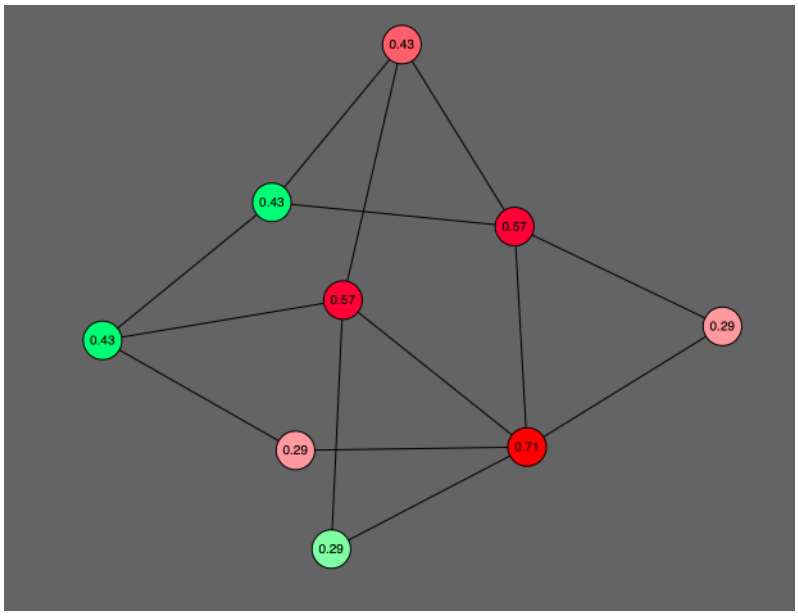


Figura 4.3: Vista de un ejemplo de grafo

En este panel se muestra también si existe algún algoritmo de difusión corriendo en ese instante con una pequeña indicación en la parte inferior izquierda.

(a) Algoritmo Pausado (b) Algoritmo "Random Walk" ejecutándose a 1 tick por secundo

Figura 4.4: Dependiendo de el algoritmo que haya seleccionado el usuario se mostrará una frase que indique el estado del mismo

4.4.2. Panel de Información

Este panel de información, dibujado en verde y situado en la parte derecha del navegador, se encarga de darle al usuario información global del grafo, de los nodos o de una arista seleccionada. Por ello dividiré esta sección en las tres vistas que puede tener este panel.

Información de un nodo

En el caso en el que el usuario pase el ratón por encima de un nodo o use el botón *"Seleccionar Nodo"* en la parte superior del panel, se mostrará la información sobre el nodo seleccionado. Esta información incluye

1.– Su **identificador** único en formato *string*.

2.– La cantidad de **información** que contiene el nodo en tiempo real en formato *float* truncado a dos decimales.

3.– El **grado** de dicho nodo, es decir, el número de aristas que lo unen a otros nodos del grafo

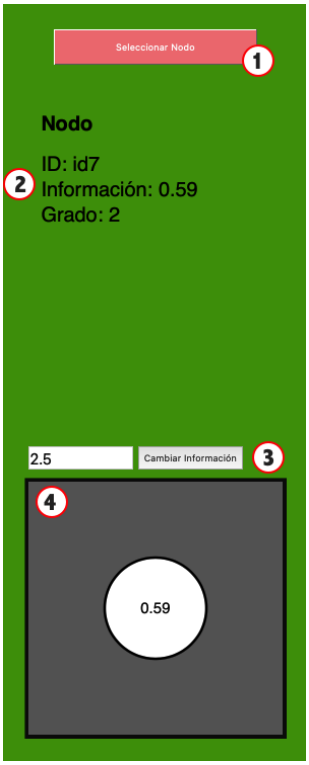


Figura 4.5: Vista del Panel de Información. (1) Botón para seleccionar un nodo. (2) Información del nodo. (3) Botón y cuadro de texto para cambiar la cantidad de información. (4) Previsualización del nodo.

Además en la parte inferior del panel hay una ventana donde se muestra una miniatura del nodo para una mejor visualización.

Por último, en este mismo panel existe la posibilidad de cambiar la información que contiene un nodo, para ello basta con seleccionarlo usando el botón *"Seleccionar Nodo"*, introducir un número decimal positivo en el cuadro de texto y confirmar pulsando el botón *"Cambiar Información"*.

Todos estos elementos pueden observarse en la figura 4.5.

Información de una arista

Si el usuario pasa el ratón por encima de una arista, ésta se coloreará de un color azul intenso en el grafo. Además, en la ventana de previsualización se mostrarán los dos nodos que une dicha arista con sus correspondientes cantidades de información.

Información del grafo

Si el usuario no pasa el ratón por encima de ningún nodo o arista, el panel de información mostrará datos globales sobre el grafo:

1.– **Número de Nodos**.

2.– **Número de Aristas** existentes en el grafo.

3.– Suma de toda la **información** existente en los nodos del grafo.

4.– El **grado medio** en el grafo, esto es una parámetro que indica el nivel de conectividad en el grafo y resulta de la división entre el número de aristas y el número de nodos.

4.4.3. Panel de Control

El segundo de los paneles disponibles para el usuario lo he denominado *Panel de Control*, ya que en éste se encuentran muchas herramientas para que sea el usuario el que modifique la estructura del grafo a su gusto. Los botones dispuestos en este panel tienen un funcionamiento muy intuintivo.

- Botones de control del grafo en su totalidad:
  - **Borrar Grafo:** Limpia el grafo actual y todos los datos que hubiera del mismo en el programa.
  - **Bloquear Grafo:** A veces puede ser interesante desactivar el algoritmo de fuerza que coloca los nodos de una forma más estética y limpia (ver sección 3.1.1). Por ello se dispone de este botón, que bloquea la posición de los nodos para que el usuario los disponga dónde desee. Los nodos bloqueados aparecerán dibujados con un círculo dorado.
- Botones para modificar los nodos:
  - **Añadir Nodo:** Si este botón está activado, cada vez que el usuario haga click dentro del panel del grafo aparecerá un nodo en la posición del ratón sin ningún tipo de información.
  - **Eliminar Nodo:** Cuando se active esta opción, cada vez que se haga click sobre cualquier nodo del

grafo, éste será eliminado, borrando todas sus aristas y eliminando la información que esté contenida en él.

- Botones para modificar aristas:
  - **Añadir Arista:** Al activar este botón, el usuario tendrá la opción de crear aristas entre los nodos. Para ello, basta con hacer click en un nodo y arrastrar el ratón hasta otro nodo con el que no se comparta ninguna arista. Se dibujará una línea en el proceso que seguirá la posición del ratón hasta que el usuario libere el ratón. Si el ratón se libera sobre otro nodo se creará la arista, en el caso en el que se libere sobre ningún nodo o con un nodo que ya esté unido, no sucederá nada.
  - **Eliminar Arista:** Con este botón seleccionado, el usuario es capaz de eliminar una o varias aristas al pulsar sobre ellas. Si el ratón está posicionado sobre la unión de dos o más aristas cuando se hace click, todas ellas serán eliminadas del grafo.

Cualquiera de los botones que se encuentran en este panel cambiará de color al ser seleccionado, indicando la opción activada. Para desactivarlos, basta con pulsar sobre el botón que ya esté seleccionado en ese momento.

Figura 4.6: Vista del Panel de Control. En este caso, la opción *"Añadir Nodo"* está seleccionada.

4.4.4. Panel de Difusión

En el último panel se encuentran las opciones para controlar los algoritmos de difusión. El primer elemento que encuentra el usario será una selección desplegable donde podrá elegir uno de los algoritmos descritos en la sección 2.3. A partir de ese momento, tenemos cuatro botones disponibles para controlar el algoritmo que hayamos seleccionado previamente: ejecutar, pausar, acelerar y revertir.

La única diferencia entre ejecutar el algoritmo o acelerarlo es la velocidad a la que se ejecuta, ya que si se pulsa el botón de acelerar éste se procesará diez veces más rápido (concretamente se ejecutará una vez cada décima de segundo). Debe usarse esta opción con precaución debido a que para grafos con muchos nodos y aristas puede suponer una ralentización de la herramienta debido a la gran cantidad de operaciones que se ejecutan por segundo.

El botón de revertir servirá para pausar el algoritmo que se esté ejecutando en ese mismo momento y además devolverá el grafo a su estado previo a la ejecución del algoritmo. Esto incluye a los nodos y aristas que se hayan añadido después de empezar el algoritmo, los cuales se perderán. También se restaurará la información que se encontraba en los nodos del grafo, que volverá también a su estado previo al algoritmo.

Por último, en este panel se dispone de la opción de añadir nodos con conexión preferencial siguiendo el modelo Barabási-Albert explicado en la sección 2.4.2. Se le pedirá al usuario que introduzca por teclado el número de nodos que desea añadir siguiendo este modelo, tras lo cual se añadirán todos los nodos secuencialmente, es decir, si el usuario decide introducir 5 nodos por ejemplo se añadirán uno a uno de tal forma que el primero también computará al introducir los siguientes, y así sucesivamente. Estos nodos tendrán un color verde tal y como se describe en la sección 3.1.2.

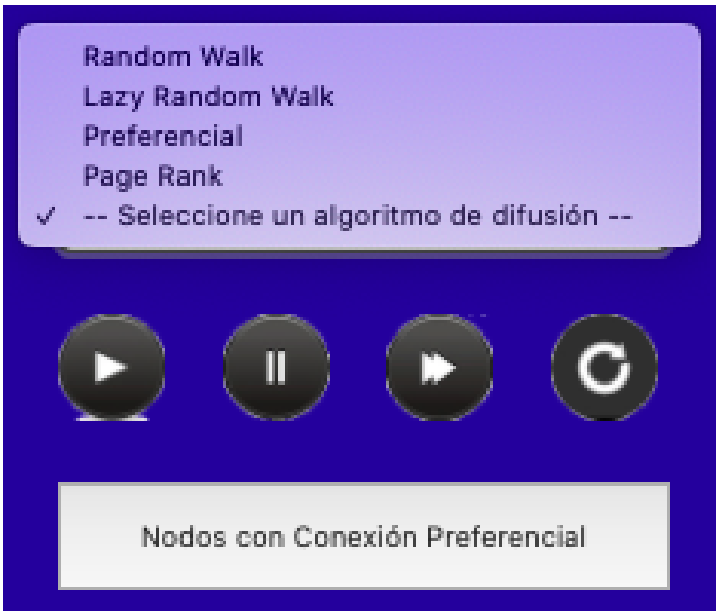


Figura 4.7: Vista del Panel de Difusión.

5

Conclusiones y Trabajo Futuro

En este capítulo daré las conclusiones finales tras la realización del trabajo tomando en cuenta la experiencia obtenida durante el mismo, además de las características que podrían seguir mejorándose en la herramienta en un futuro.

5.1. Conclusiones

El principal objetivo del proyecto era crear una herramienta didáctica con la que los estudiantes interesados en la teoría de grafos pudieran experimentar ciertos algoritmos y estructuras.

Los requerimientos y funcionalidades finales del trabajo han ido poco a poco modificándose y creciendo conforme iba evolucionando el desarrollo. Todo ello con el objetivo de hacer la herramienta más completa y versátil. Por ello no solamente tenemos un solo algoritmo de difusión si no varios, y distintas herramientas para modificar la estructura del grafo.

Finalmente se ha logrado desarrollar una herramienta bastante completa en cuento a funcionalidades pero a la vez sencilla de usar e intuitiva que podría ser usada como material docente para estudiantes de cualquier nivel que deseen explorar la teoría de grafos.

5.2. Trabajo Futuro

Dado que es una herramienta destinada a ser usada didácticamente siempre hay características que pueden ser añadidas en un futuro, como por ejemplo mayor cantidad de algoritmos con distintos criterios de difusión o incluso dejar al usuario crear sus propios criterios en tiempo de ejecucion.

Otro de los aspectos mejorables es el rendimiento general de la herramienta. Al haber tantos calculos por segundo, cuando se llega a un número elevado de nodos en el grafo dicho rendimiento puede verse afectado significativamente por lo que para grafos muy grandes sería incómodo usarla.

Una característica interesante podría ser un sistema de registro de usuarios para permitir que

cada persona que tenga una cuenta registrada pueda guardar los grafos que vaya usando, o incluso compartirlos con otros usuarios.

Por último, otro aspecto a tener en cuenta es la plataforma en la que se ejecuta la herramienta. Sería interesante adaptarla en un futuro a navegadores móviles dado que de esa forma sería mucho más accesible a un público más amplio que quisiera utilizarla de forma rápida.

Bibliografía

[1] "Graphonline, find shortest path." https://graphonline.ru/en/.

[2] "Rhumbl, easily make network visualizations." https://rhumbl.com/.

[3] "Gephi, makes graphs handy." https://gephi.org.

[4] J. Ding, O. Zeitouni, and F. Zhang, "Heat kernel for liouville brownian motion and liouville graph distance," *Communications in Mathematical Physics*, vol. 371, no. 2, pp. 561–618, 2019.

[5] A. S. S. Rao, "Population network structures, graph theory, algorithms to match subgraphs may lead to better clustering of households and communities in epidemiological studies," *Epidemiology & Infection*, vol. 148, 2020.

[6] M. Sohail and A. Irshad, "A graph theory based method to extract social structure in the society," in *Intelligent Technologies and Applications: First International Conference, INTAP 2018, Bahawalpur, Pakistan, October 23-25, 2018, Revised Selected Papers*, vol. 932, p. 437, Springer, 2019.

[7] H. Risken and T. Frank, *The Fokker-Planck Equation*. 12 2012.

[8] L. Euler, "Solutio problematis ad geometriam situs pertinentis," *Commentarii academiae scientiarum Petropolitanae*, pp. 128–140, 1741.

[9] D. A. Spielman, "Spectral graph theory." http://www.cs.yale.edu/homes/spielman/561/2009/lect08-09.pdf, 2009.

[10] A. N. Langville and C. D. Meyer, *Google's PageRank and beyond: The science of search engine rankings*. Princeton university press, 2011.

[11] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[12] J. E. Pérez, "Introducción a javascript," 2019.

[13] "p5.js." https://p5js.org.

[14] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view-controller user interface paradigm in smalltalk-80, ch. 31 (3)," *Journal of Object-Oriented Programming*, 1988.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID